R Small Group: Class 4

Amy Allen & Sara Snell

June 23, 2017

Using this document

- Code blocks and R code have a grey background (note, code nested in the text is not highlighted in the pdf version of this document but is a different font).
- # indicates a comment, and anything after a comment will not be evaluated in R
- The comments beginning with ## under the code in the grey code boxes are the output from the code directly above; any comments added by us will start with a single #
- While you can copy and paste code into R, you will learn faster if you type out the commands yourself.
- Read through the document after class. This is meant to be a reference, and ideally, you should be able to understand every line of code. If there is something you do not understand please email us with questions or ask in the following class (you're probably not the only one with the same question!).

Class 4 expectations

1. Know and understand the three basic control statements in R (for-loops, if/else statements, and while statements)
2. Learn the and/or operators for combining logical statements

Control statements

Imagine a machine that slices apples and oranges such that every slice is less than an inch thick. The machine can slice the apples without peeling them, but it needs to peel the oranges before slicing. Therefore, the machine must be told whether it is slicing an apple or an orange. If the machine receives an apple it does not have to peel before beginning to slice. However, if the machine receives an orange, it must peel the orange before slicing it.

Now think about writing the program to run this machine. You have a full pile of mixed apples and oranges. The machine can only slice one piece of fruit at a time, and you must program the machine to process all the fruit, so you must first program the machine to iterate over each piece of fruit. In programming, you use a "for-loop" to iterate over different items. The next step is to determine whether to peel the fruit. In programming, you would use an "if/else" statement, i.e. *if* the fruit is an orange, peel the fruit, *else*, do not peel the fruit. Finally, the machine must keep cutting the fruit until all the slices are less than one inch thick. In programming, you would use a "while" statement, i.e. *while* any piece of the fruit is greater than or equal to one inch, keep cutting more slices.

For-loops

Each programming language has its own syntax for the different control statements. In R, you create a for-loop according to the following basic syntax:

```
for (object in list) { expression with object }
```

When a for-loop is executed, R iterates through the given list and assigns each element of the list to the given variable, then executes the given expression for that variable. In the following example, the for-loop says iterate over `1:5` and use the `print` function to display each number.

```
for (a in 1:5) { print(a) }
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

If you run `ls` after running the for loop you see that R created a variable `a` and it has the value `5`.

```
ls()
## [1] "a"
a
## [1] 5
```

The value of `a` makes sense because the last element in the given list, `1:5`, is `5`.

You can use a for-loop to do something to every element in a list. Create a list of years, called `yrs` then print a statement saying "The year is _____". You can check the length of a vector/list with the function `length`.

```
yrs <- c("2000", "2005", "2010")
for (i in 1:length(yrs)) { print(paste("The year is", yrs[i])) }
## [1] "The year is 2000"
## [1] "The year is 2005"
## [1] "The year is 2010"
```

Traditionaly, for-loops iterate over the sequence `1...N` like above. However, R allows for-loops to iterate over any type of list. Rather than iterate over `1:length(yrs)` R can just iterate over `yrs`.

```
for (j in yrs) { print(paste("The year is", j)) }
## [1] "The year is 2000"
## [1] "The year is 2005"
## [1] "The year is 2010"
```

For loops can also be nested. For example, you can iterate over every element in a matrix.

```
mat <- matrix(1:6, nrow = 3)
mat
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
for (i in 1:nrow(mat)) {
  for (j in 1:ncol(mat)) {
    print(paste("mat[", i, ",", j, "] = ", mat[i, j], sep = ""))
  }
}
## [1] "mat[1,1] = 1"
## [1] "mat[1,2] = 4"
## [1] "mat[2,1] = 2"
## [1] "mat[2,2] = 5"
## [1] "mat[3,1] = 3"
## [1] "mat[3,2] = 6"
```

The example above iterates first by row, then by column. In other words, start with row 1 and go through every column, then move to the next row and iterate through every column, and so on.

If/else statements

In R, you create an if/else statement with the following syntax:

```
if (logical statment) { do something } else  { do something different }
```

For example:

```
if (TRUE)  { print("TRUE") } else { print("FALSE") }
## [1] "TRUE"
if (FALSE) { print("TRUE") } else { print("FALSE") }
## [1] "FALSE"
if (1 > 2) { print("TRUE") } else { print("FALSE") }
## [1] "FALSE"
```

Not all if/else statments have to have an `else` clause. By default, if no `else` statement is given R does nothing.

```
x <- 1000
if (x > 100) print("'x' is greater than 100")
## [1] "'x' is greater than 100"
```

If/else statements can also be nested:

```
y <- 25
if (y < 100) {
  if (y > 10) {
    print("10 < y < 100")
  } else {
    print("y < 10 or y > 100")
  }
} else {
  print("y < 10 or y > 100")
}
## [1] "10 < y < 100"
```

Another way to combine if/else statments is to combine the conditional statement. There are four operators for combining logical statements: `&`, `&&`, `|`, `||`. The "and" operators (`&`/`&&`) return return `TRUE` if both statments are `TRUE`. The "or" operators (`|`/`||`) return `TRUE` if either statement is `TRUE`. The The single-character operators (`&` and `|`) return a vector.

```
1 > 0 & TRUE
## [1] TRUE
FALSE | 10 ## Remember from class 1 how 10 is coerced to TRUE
## [1] TRUE
1 > 0 & c(TRUE, FALSE, FALSE, TRUE)
## [1]  TRUE FALSE FALSE  TRUE
0 > 1 & c(TRUE, FALSE, FALSE, TRUE)
## [1] FALSE FALSE FALSE FALSE
0 > 1 | c(TRUE, FALSE, FALSE, TRUE)
## [1]  TRUE FALSE FALSE  TRUE
```

The double-character operators (`&&` and `||`) will only return a single `TRUE` or `FALSE` value.

```
1 > 0 && 10 < 100
## [1] TRUE
1 > 0 && FALSE
## [1] FALSE
1 > 0 || FALSE
## [1] TRUE
1 > 0 && c(TRUE, FALSE)
## [1] TRUE
1 > 0 && c(FALSE, TRUE)
## [1] FALSE
```

Notice in the last two statments above the order of the vector matters. The double-character operators are lazy – they only check as much as they need to. If you give a double-character operator a vector, it only checks the first element.

Furthermore, if the first expression is `TRUE` the `||` operator will not even evaluate the second expression. Similarly, if the first expression is `FALSE` the `&&` operator will not evaluate the second expression.

```
TRUE  || r
## [1] TRUE
TRUE  && r
## Error in eval(expr, envir, enclos): object 'r' not found
FALSE || r
## Error in eval(expr, envir, enclos): object 'r' not found
FALSE && r
## [1] FALSE
```

You can combine logical expressions within an if/else statment (recall we defined `y` as `25` in an earlier expression).

```
if (y < 100 && y > 10) {
  print("10 < y < 100")
} else {
  print("y < 10 or y > 100")
}
## [1] "10 < y < 100"
```

Class 4 exercises

These exercises are to help you solidify and expand on the information given above. We intentionally added some concepts that were not covered above, and hope that you will take a few minutes to think through what is happening and how R is interpreting the code.

1. Create a new fruit list with 5000 fruit instead of 25, count the number of apples and the number of oranges.
2. Modify the fruit machine program to throw out the slices that are less than 0.7 inches and track the number of fruit discarded.
3. Further modify the fruit machine program to make orange slices between 0.7 and 1.0 inches and apple slices between 0.3 and 0.5 inches. Track how many oranges are discarded and how many apples are discarded.
4. Plot a histogram of the apple slice widths.
5. Plot a histogram of the number of orange slices.

Supplemental Material

While statements

The final control statement we will discuss are while statements. Although while statements are rarely used in R, they are often used in other languages and are good to understand. The while syntax in R is:

```
while (condition) {
  do something
  update condition
}
```

Try the simple example below.

```
val <- 3
while (val < 10) {
  print(val)
  val <- val + 1
}
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

If you omit the second part that updates `val` R would have printed `3` until you manually killed the process.

Combining control statements

With the control statements above you are ready to program the fruit slicing machine discussed at the beginning of class. Start by creating the fruit. (We will use the `sample` and `rnorn` functions to generate some example data).

```
fruit <- list(type = sample(x = c("orange", "apple"),
                            size = 25,
                            replace = TRUE),
              wdth = rnorm(n = 25, mean = 6, sd = 2.5))
```

You should have a list with two elements, a vector with the fruit type and a vector with the widths. Assume the peel on an orange is an eighth inch. Start with a for-loop for every piece of fruit, check if the fruit needs peeling, peel the oranges, then divide the fruit until the width of the pieces is less than one inch. Store the results in a new list called `res`.

```
## Create result list
res <- list(type = character(), pieces = numeric(), slice_width = numeric())

## For-loop iterating through the 1:(number of fruit)
for (f in 1:length(fruit$type)) {
  f_type <- fruit$type[f] ## Get the fruit type from the list
  f_wdth <- fruit$wdth[f] ## Get the fruit width from the list
```

```
  n_pieces <- 1 ## Each fruit is initially 1 piece

  ## Peel the oranges
  if (f_type == "orange") {
    f_wdth <- f_wdth - 1/8
  }

  ## Divide the fruit
  while (f_wdth >= 1) {
    f_wdth <- f_wdth/2
    n_pieces <- n_pieces*2
  }

  res$type[f] <- f_type
  res$pieces[f] <- n_pieces
  res$slice_width[f] <- f_wdth
}
```

You can now look at the results:

```
res
## $type
##  [1] "orange" "apple"  "apple"  "apple"  "apple"  "apple"  "orange"
##  [8] "orange" "apple"  "apple"  "apple"  "apple"  "orange" "apple"
## [15] "orange" "apple"  "orange" "orange" "orange" "orange" "orange"
## [22] "orange" "orange" "orange" "orange"
##
## $pieces
##  [1] 16 16  2  8  8  8  8 16  8  8  8  8  8  8  8  8 16  2  1  8  8  8
## [24] 16  8
##
## $slice_width
##  [1] 0.5047038 0.5890641 0.8908414 0.5539262 0.7557239 0.9703886 0.5321816
##  [8] 0.5028408 0.8673861 0.8469569 0.7515647 0.7382405 0.9606175 0.5947691
## [15] 0.7379360 0.7530812 0.9463348 0.5280567 0.7755894 0.3641298 0.9041165
## [22] 0.7410710 0.7851768 0.5612116 0.9789930
```